Toward a framework for ontology modularity

Zubeida C. Khan
Council for Scientific and Industrial Research
Pretoria
South Africa
zkhan@csir.co.za

C. Maria Keet
Department of Computer Science
University of Cape Town
South Africa
mkeet@cs.uct.ac.za

ABSTRACT

Dividing up data or information into smaller components—modules—is a well-know approach to a range of problems, such as scalability and model comprehension. The use of modules in ontologies at the knowledge layer is receiving increased attention, and a plethora of approaches, algorithms, and tools exist, which, however, yield only very limited success. This is mainly because wrong combinations of techniques are being used. To solve this issue, we examine the modules' use-cases, types, techniques, and properties from the literature. This is used to create a framework for ontology modularity, such that a user with a certain use case will know the type of modules needed, and therewith then also the appropriate technique to realise it and what properties the resultant modules will have. This framework is then evaluated with three case studies, begin the QUDT, FMA, and OpenGalen ontologies.

Categories and Subject Descriptors

H.4 [**Ontology**]: Ontology Modularity Ontology Modularisation; D.2.8 [**Theoretical framework**]: Characteristics—*use cases, types, techniques, properties, evaluation metrics*

Keywords

Ontologies, Modularity

1. INTRODUCTION

Modularity is an ideal solution for dealing with cognitive overload for both machines and human, as it eases computational complexity, and simplifies comprehension and understanding by offering smaller, suitably-sized subsets of an ontology. An increase in the use of modularity for dealing with large ontologies has resulted in a plethora of approaches, and tools in the field. While there exists some literature that provide information about certain criteria of modularity, there is a lack of well-defined foundational aspects for ontology modularity. It is unclear which evaluation metrics are to be considered for different module types, what type of modules different techniques produce, and so on. This uncertainty is reinforced in existing work by d'Aquin et al. [7] where it was found that indeed the evaluation of a modularisation depends on an application's requirements and there is no universal modularisation and that a formal, well described framework, or theory for modularity is lacking.

The lack of such a theory opens up a number of issues and unanswered questions. The issues that ontology developers face include difficulty in modularity technique selection, and insufficient modularity tools for scenarios. Several modularity techniques and tools exist but it is not clear which one should be applied for a particular scenario. For instance, in modularising the data mining DMOP ontology, several modularisation tools were considered but resulted in modules that were too large to use [19]. Furthermore, existing techniques are not sufficient in creating compact modules when source ontologies exhibit tight coupling with many relational properties [20]. Unanswered questions include the following:

1. Given that we wish to create an ontology module with a certain purpose or *use-case* in mind, which modularity *type* of module could this result in?
2. If we wish to create a module of a certain *type*, which is the best *technique* to use?
3. By using a particular *technique*, which *properties* will the resultant module exhibit?

Existing literature on modularity provide disparate information about such aspects of modularity but an explicit and comprehensive list of the underlying dimensions of modularity is lacking. In this paper, we present a framework for ontology modularity, by firstly 1) identifying the core dimensions that should be taken into consideration for ontology modularity, 2) populating the dimensions with criteria, and 3) creating relations between the criteria that reveal dependencies and, hence, suggestions on which parameters go well together and which do not.

The results reveal that the framework can be used to guide the modularisation process, and it offers new insights for annotating ontology modules for improved metadata. The dependencies of the framework can be used to solve the modularity technique selection issue and to answer all the proposed questions. In addition, the tool issue concerning the current state of ontology modularity has been refined to reveal that existing tools are not sufficient nor maintained, and that there is a heavy reliance on manual methods as a technique.

The remainder of the paper is structures as follows. We

SAICSIT '15, September 28-30, 2015, Stellenbosch, South Africa
 ISBN 978-1-4503-3683-3/15/09…$15.00
DOI: http://dx.doi.org/10.1145/2815782.2815819

describe the ontology modules' dimensions in Section 2, which is followed by the framework in Section 3. The two case studies are described in Section 4, followed by a discussion in Section 5. Lastly, we conclude in Section 6.

2. DIMENSIONS

In this section we describe each module dimension and populate them by identifying respective sub-dimensions for them. Identifying these dimensions will assist with answering the questions posed in Section 1. The five dimensions concerning modularity that we identified are: use-case, type, technique, property, and evaluation metric.

2.1 Use cases

The type of module that is created greatly depends on the reason or purpose for which modularity was considered; having different purposes for modularity can lead to different modules. There are many use-cases for creating and using ontology modules which we identify and discuss in this section. The different types of use-cases, also referred to as purposes, goals, benefits or rationale for modularity has been mentioned in numerous works [6, 7, 24, 29, 31].

Maintenance. Ontologies are ever-changing. As such, there is a need for constant updates and maintenance. Enormous monolithic ontologies cannot be easily maintained, all at once. It is a task that is prone to error and omission. The division of and ontology into modules can assist with facilitating the maintenance of large and highly complex ontologies. A set of modules could be easily managed, as it is divided into smaller subsets. Not all the modules in a system need to be modified if there is change in the ontology thus the knowledge update is localised within the relevant module(s). Maintenance also promotes collaborative efforts, which is discussed as a use-case in a subsequent paragraph.

Reasoning. Ontology reasoners do not perform well when the ontologies in question are large and complex [32]. The performance of the reasoner decreases as the size of the ontology increases. Large ontologies cause reasoners to use too much of time and memory and often crash and 'die'. Consequently, reasoners will perform better with regard to efficiency if there is less knowledge to infer. In addition, it will only be necessary to reason over modules that have been evolved since the last reasoning task. The data mining optimisation ontology (DMOP), for instance, is not vert efficient upon reasoning, and has a classification time of ± 10 minutes [19].

Validation. Ontology validation is when developers analyse the ontology to determine whether it contains anomalies, and whether it meets user requirements. It is rather difficult for a single expert to validate and understand the model as a whole in a large ontology [32]. In order to solve this, the ontology could be modularised to a size that is comprehensible by a human. Naturally, smaller modules are easier for a human to understand which will aid in validation. Identifying errors such as inconsistency and redundancy, and guaranteeing that the ontology meets all the functional requirements in large, monolithic ontologies is a difficult process.

Processing. Ontology related tools such as engineering, mediation, metrics, and editor tools do not perform well when processing large ontologies [1, 2, 25]. For the NCI cancer ontology [13], the BioPortal [33] visualisation tool takes several minutes to load large taxonomy branches of an ontology, using a machine with an Intel Core 2 Duo Processor with 4GB of RAM. Using the OWL metrics tool[1] to compute the metrics of the NCI cancer ontology took 12 minutes to process before it returned an ontology parsing error. These types of scalability issues are a challenge for developers when using these large ontologies. Since smaller ontologies take a shorter time to open, load, and use with tools, having smaller interrelated modules instead of large and complex ontologies could possibly improve the performance of the processing tools.

Comprehension. It causes confusion when humans are required to understand and use ontologies with thousands of terms. These large ontologies are beyond our capability to properly comprehend. Keet proposes the use of abstraction by removing some knowledge from the system to assist with ontology comprehension [18]. Domain experts without skills and expertise in logic sometimes rely on visual ontology engineering tools for creating ontologies. Visual ontology engineering tools aids with ontology development but have processing difficulties with large ontologies.

Comprehension differs from validation for modules, in that for validation, all components of the ontology need to be considered. However, for comprehension, or for human understanding purposes, simpler views omitting unnecessary components can be considered.

Collaborative efforts. Collaborative efforts in ontology development allows a number of experts to combine their knowledge towards the goal of creating an ontology. Using ontology modules enables the division of work tasks. In order to avoid conflict between different versions of the ontology by different developers, it is sensible to divide the ontology to different modules and allow specific people to create and modify specific modules without altering the entire system. This also promotes the parallelisation of ontology development.

Reuse. At times, developers only require a subset of terms from an ontology and not the entire system to reuse in another ontology. For instance, in the Subcellular Anatomy Ontology (SAO) ontology [21], there only exist 3D entities yet they import the entire BFO ontology with both 3D and 4D entities. The smaller BFO-Continuant ontology of the ROMULUS repository which only contains 3D entities [20] could be used instead. Thus modular ontologies provide infrastructure for ontology development whereby ontology modules can be easily extracted and reused. The modular components can then be easily adapted for the application at hand.

2.2 Properties

Modules have properties that are associated with them. In this section we identify and describe the properties that modules exhibit. Properties exist in isolation in a single module, and also in sets of related modules.

2.2.1 Properties of a module

In this section, we describe properties that a module exhibits by itself.

Seed signature describes cases where the user specifies some entities to base the resulting module on [5, 8, 9]. This could be done by identifying all the heterogeneous domains in order to divide a large ontology, or to identify a single entity in an ontology to expand or extract modules.

[1]http://mowl-power.cs.man.ac.uk:8080/metrics/owlmetrics

Information removal occurs when specific parts of the ontology are selected to be removed from the ontology, resulting in a module without all the detail of the original ontology.

Abstraction is the property of hiding undesirable information from an ontology at different levels [12, 17]. It is used as a principal to give the user a simplified view of the ontology for comprehension. To do this, for different modules in the system, there exists information with more or less detail. However, the source ontology with all the original information still exists in the system as a related module.

> ***Breadth abstraction*** occurs in an ontology to provide a simpler view of the structure of the ontology, where some relational properties of entities in the module are removed. Hence the 'breadth' of the ontology is reduced.
>
> ***Depth abstraction*** occurs in an ontology to provide a higher level view of the ontology, where lower-level classes are removed. Hence the 'depth' of the ontology is reduced.

Refinement occurs in ontology modules where some new axioms are added to the module, to assist with inter-module links, or when computationally-expensive ontology language features are modified resulting in new axioms. For instance, to improve the reasoning time for the DMOP ontology, the `InverseObjectProperties` axiom was removed. It was, however, replaced with axioms containting the OWL `ObjectInverseOf` axiom [19].

Stand-alone describes a module that has no connections to or imports with other ontologies.

Source ontology describes cases where there is an original ontology that is to be modularised.

Proper subset occurs in a module that contains a set of entities that are contained in a source ontology whereby the module has fewer entities than the source ontology.

Imports is the property that a module that itself is composed of other components, by using the `owl:import` statement declared for importing another ontology.

2.2.2 Properties of a set of related modules

In this section we describe the properties that a set of modules exhibit altogether, and in relation to one another.

Partitioning occurs in large, complex ontology whereby a source ontology O is structurally divided or decomposed into a set of modules $M_a, ..., M_z$, thereby allowing concurrent reuse in distributed systems [7, 10].

Overlapping refers to cases where entities in an ontology system can be found in more than one module of the system [23]. If A and B are two ontology modules that are overlapping, this means either that A includes some entities of B, B includes some entities of A or both. These modules partially cover the same entities and may have dependencies on one another. Thus changes to one module in a system may affect some other modules.

Mutual exclusion (or disjointness) in modules occurs when entities in an ontology system are not found in more than one module of the system [23]. If A and B are two ontologies that are mutually exclusive, this means that A excludes all entities of B and B excludes all entities of A.

Union Equivalence occurs when the union of a set of modules is semantically equivalent to the original ontology.

Inter-module interaction occurs in modules that have external links to other modules in a system. Ontology modules in a system require mechanisms to relate entities in a similar way to their original existence in the monolithic module to ensure that the meaning of the source ontology is preserved. Classes from different modules may require equivalence or subsumption relations, including the use of roles to interconnect entities. For instance, in a set of related modules, there is a module of Animals and a module of Habitats. In the Animals module, there is an entity `Fish`. In the Habitats module, there is an entity `WaterHabitat`. Both these entities are to be connected or linked by the `livesIn` object property.

Pre-assigned number of modules occurs when the number of modules to be created for a set is known and specified by the ontology developer prior to development.

2.3 Types

We propose that ontology modules can be classified into different types, based on the nature of the module. Borgo classifies modules as: 1) modules for a single ontology, those modules which aid with organising and managing domain coverage, 2) modules for several ontologies, basic functionality that when combined lead to better quality ontologies, and 3) modules for everything, which has several different meanings such isolating/developing branches of a taxonomy, collecting categories according to a domain, isolating patterns, and more [3]. From existing ontology modules and Borgo's research [3], we have identified and refined modularity type dimensions as subtypes which are classified into four main types of modules: structural, functional, abstraction, and expressiveness modules. Understanding the type of a particular module is of interest towards creating a foundation for ontology modularity.

Besides using and refining Borgo's modularity dimensions as modules subtypes, we have identified new subtypes: locality, privacy, axiom abstraction, type abstraction, high-level abstraction, weighted abstraction, expressiveness sub-language, and expressiveness feature modules which are further explained in the following sections.

2.3.1 Functional modules

Functional modules are those in which function is the driving force. For these type of modules, the users identify the functional components or subject domains within an ontology to be separately modularised. This assists with selective reuse of an ontology. We now describe the subtypes that exist for functional modules.

Privacy modules are those in which certain information must be hidden or removed from an ontology so that modules can be kept private from each other.

Ontology design pattern modules describe those for which ontology is to be modularised by identifying a part of the ontology for reuse as a best practice to solve recurring ontology issues, Hence, one can identify and isolate a new ontology design pattern for general reuse.

Subject domain modules are those modules that are created when an ontology is subdivided according to the subject domains present in the ontology. For instance, a large ontology of food could be modularised into separate modules of food, vegetables, meat, etc.

Isolation branch modules are modules where a specific subset of entities (a signature) from an ontology is extracted. However, entities that have weak dependencies to the signature are not to be included in the module hence we

have an 'isolation branch'. For instance, to isolate the 'endurant' branch of DOLCE, the `dolce:physical-endurant` entity is a direct subclass of `dolce:endurant` to include. `dolce:perdurant` must be excluded, because it is linked to `dolce:endurant` in terms of participation.

Locality modules are modules where a specific subset of entities (a signature) from an ontology is extracted. However, every single entity and axiom that is dependent on the subset is included in the module. For the previous example, this means that the `dolce:perdurant` entity is to be included in the module, along with every entity that is related to it.

2.3.2 Structural modules

Structural modules are those ontologies that have been partitioned into modules based on structure and hierarchy by using the ontology graph. In this case each module is to be separate from one another; the ideal goal is to have disjoint modules so that the union of all modules is equivalent to the original ontology. We now describe the subtypes that exist for structural modules.

Domain coverage modules are created when there exists a large monolithic ontology, and the ontology is divided structurally to facilitate easy ontology engineering and maintenance.

Ontology matching modules are created when an ontology must be modularised in order to assist with ontology matching. Here, the ontology is partitioned into disjoint modules so that there is no repetition of entities when matching occurs.

Optimal reasoning modules are created when an ontology is large and must be divided to smaller reasonably-sized modules to assist with overall reasoning of the ontology and to ensure that reasoners do not malfunction.

2.3.3 Abstraction modules

For abstraction modules, some detail is to be hidden from the ontology. This creates a simplified view of the ontology making it 'lightweight' with less detail. We now describe the subtypes that exist for abstraction modules.

Axiom abstraction modules are created when one wishes to create a module having fewer axioms between entities that exist in the original ontology, thereby decreasing the horizontal structure of the ontology.

Entity type abstraction modules are created when one wishes to remove a certain type of entity from the ontology, e.g., data properties or object properties.

High-level abstraction modules are created when one wishes to create a module where only higher-level classes of the ontology are required, and lower-level entities are deleted. This decreases the vertical structure of the ontology.

Weighted abstraction modules are created when the user decides on entities in the ontology that are more important than others to be included in the module.

2.3.4 Expressiveness modules

For expressiveness modules, an ontology is modularised in terms of expressive power by slimming down to a specific ontology sub-language species or by removing some language features. We now describe the subtypes that exist for expressiveness modules

Expressiveness sub-language modules are created when one wishes to have modules of an ontology where the axioms in the module are slimmed down to a sub-language of a core ontology language. For instance, the sub-languages OWL 2 EL, OWL 2 QL and OWL 2 RL of the OWL 2 ontology language. This could be for efficiency or reasoning purposes. For instance, TopSPIN supports OWL 2 RL reasoning.

Expressiveness feature modules are created when one wishes to have modules where the axioms in the module are slimmed down to contain limited language features but are not necessarily contained in a sub-language. For instance, one may only require concept subsumption and conjunction features in a module for efficiency or reasoning purposes.

By refining Borgo's modularity dimensions and analysing existing modules, in this section we have identified four main module types: structural, functional, abstraction, and expressiveness, and each of these have a number of subtypes.

2.4 Techniques

In this section, we identify possible techniques from existing approaches that may be used to create modules, and classify them into categories.

Graph theory approaches are those that have been designed to be applied to the general problem of community detection. In graphs, communities are clusters of nodes that are fairly independent of each other with weak links between them. Applying graph theory techniques to ontologies deals with modularising them according to structure in order to identify modules together with the inter-module links. The advantage of such an approach would be that if there is already a set of modules, algorithms can be applied to improve the modules until the best set of modules has been generated. The disadvantage of such an approach is that the final ontology strongly depends on the initial partitioning and if there is no information about how the initial partitioning ought to be performed, the method performs poorly.

Graph partitioning is when a large graph is divided into partitions. The vertices are not shared across different partitions, and the number of partitions is known. In terms of ontologies, graph partitioning algorithms could be used for the structural division of the ontology for it would generate modules of equal size. Also, the user must know the number of modules that the ontology should be modularised into.

In the PATO partitioning tool [29], it is performed by using maximal line islands to generate partitions in the graph. A maximal line island checks that for a set of entities, the connection between the entities inside the set is higher than the connection to entities outside the set. Unlike traditional graph partitioning, in PATO, the number of partitions to be created is unknown prior to the modularisation.

Modularity maximisation methods aim at optimising the connection between nodes in graphs and to do so the modularity function Q measures the concentration of edges within modules compared with random distribution of links between all nodes. In terms of ontologies, this means that the emphasis for creating modules is not based on the location of concepts but rather on the strength of axiomatic relations that concepts have with others. To date, there has been one application of using modularity maximisation to perform ontology modularity [11].

Statistical approaches emphasise on using statistical equations to create ontology modules by converting the entities of an ontology to data points. After converting the

entities to data points, statistical methods and functions are applied onto the data to create modules. Thus, the ontology is viewed as a data set in order to modularise.

Hierarchical clustering is used for grouping data, in cases where little is known about the data, such as the number of partitions it should be split to. Hierarchical clustering methods are either agglomerative or divisive. An agglomerative strategy is one in which each data point is placed in separate clusters, which are then merged based on a given distance function between data points in clusters. For divisive strategies, the data is divided recursively as one traverses down the hierarchy.

To date, there has been one application of using hierarchical clustering to perform ontology modularity [11]. It was found that the two hierarchical clustering algorithms obtained similar results when compared to other graph theory approaches. However, semantically, other approaches worked better, for performing modularisation in the pizza ontology because their modules created seperate modules for vegetarian, non-vegetarian, and general pizza entities while the modules of the hierarchical approach did not.

2.4.1 *Semantic approaches*

In the previous section, the structure of the ontology was the driving force for modularisation. In this section, the entities and axioms of the ontology are used for the modularity approach. These approaches are user driven, ie., a user provides some initial information about entities to carry out the modularisation process.

Locality modularity is used to generate modules based on a given signature such that conservation extension holds for the given module. Conservation extension is the notion that every axiom's meaning from the original ontology is preserved in the module. Conservation extension is greatly influenced by the atomic structure of the ontology. In an ontology, an atom is a set of axioms that depend on each other. For instance, if one were to generate a locality module of endurant entities (whole objects) from the DOLCE ontology, a number of perdurant entities (entities unfolding in time e.g., processes) would also be contained in the module, because there exists an axiom `endurant participant-in some perdurant`, in the DOLCE ontology. Therefore the module would not be restricted only to endurant entities because the conservation extension of the original ontology is guaranteed. Biological ontologies from the BioPortal repository have had great success for modularisation using locality methods [9], thanks to them having on average of 2 axioms per atom.

Query-based modularity approaches require that the user initially creates a query with certain conditions in a query language such as SPARQL to automaticaly generate a module. Noy and Musen [22] use queries generate ontology views. This type of modularity depends heavily on user input as the user decides, at every step, which entities to be included in the ontology.

The segmentation approach to query-based modularity exploits semantic links between ontology entities to extract relevant segments of an ontology, based on an input entity [30]. It does not include sibling classes in the module. An evaluation on this segmentation approach reveals that the large GALEN ontology of medical terminology was reduced by a factor of 20 [30].

Semantic-based abstraction has been applied to ORM conceptual models [4, 17], whereby the semantics of the model is analysed using pre-defined rules to determine entities to include in the module; these are deemed more important while others are removed. This could be applied to ontologies by designing a set of weighted rules to guide the ontological abstraction process. For instance, a weighting that deems a class with multiple existentially quantified more important than entities without this hence removing them.

An* a priori *modularity method is one in which the modules of the domain is decided at the onset of ontology development.

Manual modularity methods apply when the ontology developer drives the entire process, without automatic tool usage, i.e., the ontology developer decides which entities and axioms should be removed and manually creates a module based on this.

2.5 Evaluation metrics

The evaluation of modularity techniques using metrics is an important process to measure the quality of ontology modules. Existing studies [6, 7, 28] mention a number of metric techniques such as size, logical correctness, cohesion etc. Populating the evaluation metrics with sub-dimensions is intended for future work as it additionally requires an application component with which to quantitatively measure ontology modules.

3. FRAMEWORK FOR ONTOLOGY MODULARITY

We formulate that use-cases are related to types, types are related to techniques, and techniques are related to properties. A high-level view of the framework is displayed in Fig 1.

Figure 1: A high-level view of the framework.

3.1 Dependencies among characteristics

An experiment was conducted in which 176 modules were collected and classified according to the modularity characteristics. The full experimental set-up, with data files can be accessed online[2]. In this section, we use the results of the experiment to create dependencies among the modular characteristics.

The dependencies are used to answer the questions about modularity described in Section 1, and will result in the creation of a framework for ontology modularity. The answers to the question follow in the subsequent sections and diagrams. For instance, regarding the first question, if we wish to create an ontology for the use-case of comprehension, this could result in an abstraction type module.

3.1.1 *Dependencies between use-case ↔ type*

We, firstly examine the relationship between use-cases and types of module to determine which types of modules the use-cases result in. When maintenance is the use-case, this

[2]www.thezfiles.co.za/Modules/testfiles.zip

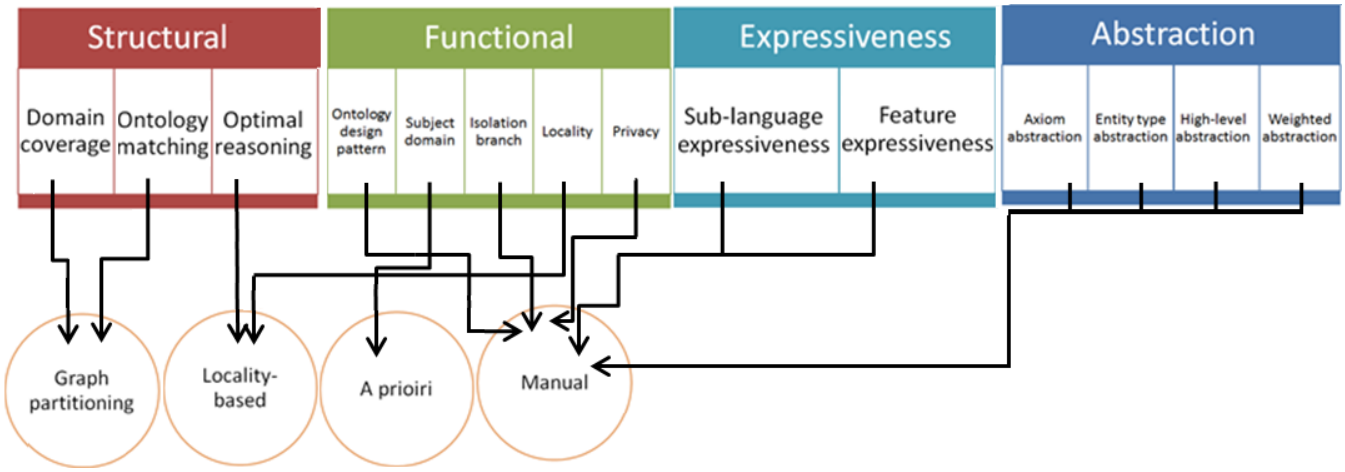

Figure 2: The dependencies between module types and techniques.

results in subject-domain modules or domain coverage modules. Hence maintenance results in some type of functional or structural modules. Reasoning results in optimal reasoning modules which are a kind of structural module, or expressiveness sub-language or expressiveness feature modules which are both of the broader expressiveness type.

The validation use-case results in the same modules as the maintenance use-case, i.e., subject-domain modules or domain coverage modules. Hence validation results in some type of functional or structural modules. The processing use-case results in ontology matching or optimal reasoning modules which are both structural modules. When comprehension is the use-case at hand, the modules are either axiom abstraction, entity type abstraction, high-level abstraction or weighted abstraction modules. These four modules are of a broader expressiveness type.

The collaboration use-case results in subject domain, privacy, domain coverage, or ontology matching modules. Subject domain, and privacy modules are of broader type structural. Domain coverage, and ontology matching modules are a functional type. For the reuse use-case, we get ontology design pattern, subject domain, isolation branch, locality, and privacy modules. These are functional type modules.

3.1.2 *Dependencies between type ↔ technique*

Next, an ontology developer may be interested in which is the best technique to use for creating a particular type of ontology module. For creating a ontology design pattern, isolation branch, and privacy modules,thus far there is only evidence of manual methods being used. For subject domain modules, when a large ontology must be subdivided according to specific subject domains, *a prioiri* modularity techniques are used. When creating locality-based modules, then naturally locality modularisation approaches are used. Thus the three types of techniques employed for creating structural modules in general are locality, *a prioiri*, and manual methods.

When the modules to be developed are those for domain coverage, or ontology matching, graph partitioning techniques are used. For optimal reasoning modules, i.e., when an ontology is extremely large and must be divided to smaller reasonably-sized modules to assist with reasoning, graph partitioning is used. Thus, for creating structural modules, graph partitioning or locality-based methods are used.

For all abstraction modules, i.e., axiom, entity type, high-level, or weighted abstraction modules, only manual methods are used. For all expressiveness modules, i.e., sub-language or feature expressiveness modules, manual methods are used. The dependencies between these types and techniques are displayed in Fig. 2.

3.1.3 *Dependencies between technique ↔ property*

Next, we examine which technique results in modules with a particular property. When graph partitioning techniques are used, the resulting modules have the following properties: information retrieval, stand-alone, source ontology, or proper subset. Since graph partitioning always results in a set of related modules, the properties of the set that exist for such modules are overlapping, mutual exclusion, partitioning, or inter-module interaction. Locality-based methods results in modules with the following properties: seed signature, information retrieval, stand-alone, source ontology, or proper subset. Locality-based methods have also been used to create a set of inter-related modules so its modules exhibits the set property of overlapping.

The *a prioiri* methods result in modules with these properties: stand-alone, or imports. *A priori* modules are also created a set so there are properties for the set as a whole which are overlapping, or pre-assigned number of modules. For manual methods, the resulting modules exhibit the following properties: seed signature, information removal, abstraction, breadth abstraction, depth abstraction, refinement, stand-alone, source ontology, proper subset, or imports. The dependency relationship between all these techniques and properties are displayed in Fig. 3.

4. EVALUATION: CASE STUDIES

In this section, we test the framework to guide the modularisation process. We select three case studies of ontology modules. These modules were not from the 'training' set of modules that were used in the experimental evaluation to create the framework. We consider these modules as the 'testing' set of modules.

QUDT ontology modules.

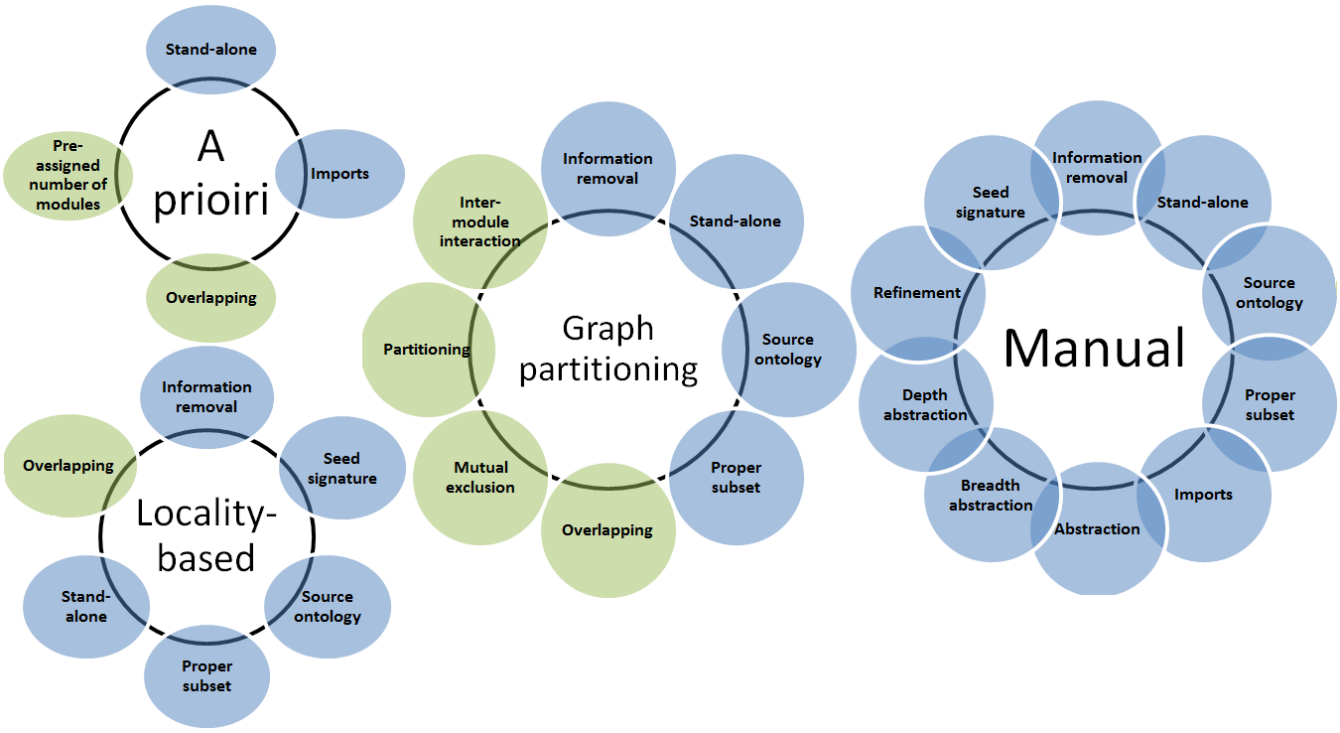

Figure 3: The dependencies between techniques and properties.

The Quantities, Units, Dimensions and Data Types (QUDT) ontologies are a set of modules focused on terminology used in science and engineering for representing physical quantities, units of measure, and their dimensions [14].

Use-case Identify the use-case for the modularity of the set of ontologies. In the set of seven QUDT modules, there is a total of 4067 entities, hence it is a large domain and is divided into several modules to facilitate maintenance and validation. One of the goals outlined in the QUDT specification states that parts of the vocabulary will be of interest to certain users or applications depending on the use-case. Hence the individual modules in the set could be used for reuse. Lastly, the modular approach of the subject domain means that a team of experts could work with specific modules thereby enabling collaborative efforts. Hence the four use-cases for the QUDT ontology modules are maintenance, validation, reuse, and collaborative efforts.

Type Since the use-case(s) have been identified, we can now refer to the framework to check for the next step of the modularisation process. The framework states that use-cases result in module types. According to the dependencies, when maintenance or validation is the use-case, this results in subject-domain or domain coverage modules. Collaborative efforts result in subject domain, privacy, domain coverage, or ontology matching modules. For reuse, the resulting module(s) is ontology design pattern, subject domain, isolation branch, locality or privacy modules. Across all four use-case to type dependencies, the common module type that maintenance, validation, collaborative efforts, and reuse result in is the subject domain modules. Hence the set of QUDT modules are a set of subject domain modules, whereby a large ontology is divided according to the subject domains within the ontology.

Technique The type of module drives the modularisation technique. Given that the module is a subject domain module, according to the framework, such modules are created using the *a prioiri* technique. Hence, the modules to be created are decided at the onset of ontology development. There is no source ontology for QUDT containing the entire domain, there is only a set of modules hence we can assume that a modular approach was decided at the onset of ontology development.

Property Lastly, by using an *a priori* technique, which module properties can we expect of the QUDT set of modules? The framework states that *a priori* techniques result in modules each with the stand-alone or imports properties, and as a set of modules, they exhibit the overlapping or pre-assigned number of modules properties. Two of the QUDT modules exhibit the stand-alone property only. The remaining six modules each exhibit the imports property only. As a set of modules, the modules exhibit the following set properties: overlapping and pre-assigned number of modules.

A summary of the case-study for the QUDT modules is shown in Fig. 4.

Foundational Model of Anatomy module.

The Foundational Model of Anatomy Ontology (FMA) is a reference ontology for the domain of human anatomy [27].

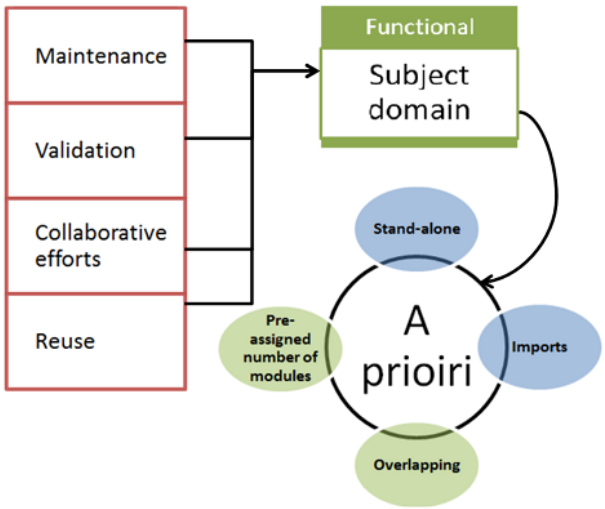

Figure 4: The dependencies between the module dimensions for the QUDT modules.

Use-case We wish to create a module with a small selection of knowledge from the FMA ontology, particularly to be reused in the creation of an ontology about infection. For our proposed ontology about infection, we require all the information about body substances from the FMA ontology to be reused.

Type Given that the identified use-case for the module is reuse, we refer to the framework to check which type of modules result from reuse. The reuse technique results in all the functional module types: ontology design pattern, subject domain modules, isolation branch, locality, and privacy modules. Since we wish to extract a subset of the FMA ontology, we consider the creation of either an isolation branch module, or a locality module. Looking at the entities about body substances in the FMA ontology, we realise that we do not wish to preserve entities with weak dependencies or relations to the body substances entities. Hence, we decide to create an isolation branch type module.

Technique For the isolation branch type module, the only technique that can be used to create such a module is manual methods. To create a branch module of body substances from the FMA ontology, we delete all entities besides the taxonomic branches referring to body substances entities.

Property The framework states that when manual methods are used, the resulting module could have the following properties: seed signature, information removal, abstraction, breadth abstraction, depth abstraction, refinement, stand-alone, source ontology, proper subset, or imports. The FMA body substances module exhibits the following properties: seed signature, information removal, stand-alone, source ontology, and proper subset.

OpenGalen EL module.

The OpenGalen ontology [26] is a common reference module for application-independent and language-independent model of medical concepts.

Use-case ELK [16] is a reasoner for the lightweight ontology language OWL EL created for improved reasoning for large ontologies. A study on the evaluation of the ELK reasoner requires modules for the use-case of reasoning.

Type Since the use-case is reasoning, this could result in the following types: optimal reasoning modules, expressiveness sub-language or expressiveness feature modules. In order to test out the ELK reasoner, the development team created an EL version of the OpenGalen ontology[3]. Hence the module type is an expressiveness sub-language module.

Technique Thus far, the technique used for creating expressiveness sub-language modules is manual methods. The study on the evaluation of the ELK reasoner states that an EL version of the OpenGALEN ontology was created by removing `InverseObjectProperties` and `FunctionalObjectProperties` axioms, hence we can assume that manual methods were used for this.

Property The framework states that when manual methods are used, the resulting module could have the following properties: seed signature, information removal, abstraction, breadth abstraction, depth abstraction, refinement, stand-alone, source ontology, proper subset, or imports. The OpenGalen EL module exhibits the following properties: information removal, stand-alone, source ontology, and proper subset.

5. DISCUSSION

The modularity dimensions led to the ontology modularity framework which can be used to answer the earlier proposed questions by using the dependency relations among modularity dimensions. Furthermore, the framework can be used to solve the first issue concerning modularity: ontology developers face difficulty in modularity technique selection, by referring to the framework to check which combination of technique results from the purpose or use-case of the module. For the latter problem, that there are insufficient tools for modularity, this issue has not been solved yet but has been refined it as follows.

There are currently not enough automated tools for all the modularity techniques. There is no tool to implement modularity maximisation, semantic-based abstraction, and hierarchical clustering. For the tools that are available, they are hardly maintained. We had hoped to include enormous ontologies that were partitioned into a set of modules using partitioning tools. However, SWOOP partitioning tool [15] could not be used for such large ontologies. SWOOP could not open the FMA ontology, due to java memory errors despite manually changing the java heap space parameters. For creating query-based modules with PROMPT traversal views, it malfunctioned and returned a null pointer exception.

For creating ontology design patterns, isolation branch, privacy, sub-language expressiveness, feature expressiveness, axiom abstraction, entity type abstraction, high level abstraction, and weighted abstraction modules, manual methods are used. Implementing tool-based methods for at least

[3]http://code.google.com/p/elk-reasoner/wiki/TestOntologies

some of the abstraction and expressiveness type modules is within reach, given the advancements in ontology tools such as the OWL API.

The results from the evaluation of the framework using a new set of 'testing' ontologies of the case-studies demonstrate that the framework is indeed useful for guiding the modularisation process. For the QUDT and OpenGalen modules that already existed, the use-cases and techniques of them correspond with the dimensions that are obtained by using the framework for the modularisation process thereby ensuring the framework's correctness in usability. In addition, the framework provides the ontology developer with an idea of which properties the module should exhibit, and also classifies the module by identifying its type. Such criteria can assist in annotating ontology modules towards improved metadata thereby promoting ontology reuse. For the FMA ontology, where a new module had to be created, the framework assisted in guiding the entire process.

6. CONCLUSION

We have identified and defined dimensions concerning modularity. These dimensions were used in an experimental evaluation with a set of 176 'training' ontology modules resulting in dependencies among the modularity dimensions. This led to the creation of a framework for ontology modularity. The framework can be used to answer a developer's questions about what to do when creating an ontology module—hithertho not possible in an informed way—starting from a use-case, solve the issue concerning modularity technique selection, and refine the other issue concerning insufficient modularisation tools.

The issue of insufficient modularisation tools is a point for further investigation. Evaluation metrics for modules are still in their infancy, as are ontology metrics in general, but it would be nice to have as a fully functional additional dimension. This would assist in measuring the quality of ontology modules.

7. REFERENCES

[1] E. Antezana, M. Egaña, W. Blondè, A. Illarramendi, I. n. Bilbao, B. De Baets, R. Stevens, V. Mironov, and M. Kuiper. The cell cycle ontology: an application ontology for the representation and integrated analysis of the cell cycle process. *Genome Biology*, 10(5):R58, 2009.

[2] K. T. Belloze, D. I. S. B. Monteiro, T. F. Lima, F. P. S. Jr., and M. C. R. Cavalcanti. An evaluation of annotation tools for biomedical texts. In *Joint V Seminar on Ontology Research in Brazil and VII International Workshop on Metamodels, Ontologies and Semantic Technologies*, volume 938 of *CEUR Workshop Proc.* CEUR-WS.org, 2012. Recife, Brazil, September 19-21.

[3] S. Borgo. Goals of modularity: A voice from the foundational viewpoint. In O. Kutz and T. Schneider, editors, *Fifth International Workshop on Modular Ontologies (WOMO'11)*, volume 230 of *Frontiers in Artificial Intelligence and Applications*, pages 1–6. IOS Press, 2011. Ljubljana, Slovenia, August.

[4] L. J. Campbell, T. A. Halpin, and H. A. Proper. Conceptual schemas with abstractions: Making flat conceptual schemas more comprehensible. *Data Knowledge Engineering*, 20(1):39–85, 1996.

[5] B. Cuenca Grau, I. Horrocks, Y. Kazakov, and U. Sattler. Modular Reuse of Ontologies: Theory and Practice. *Journal of Artificial Intelligence Research*, 31:273–318, 2008.

[6] M. d'Aquin, A. Schlicht, H. Stuckenschmidt, and M. Sabou. Ontology modularization for knowledge selection: Experiments and evaluations. In R. Wagner, N. Revell, and G. Pernul, editors, *18th International Conference on Database and Expert Systems Applications (DEXA'07)*, volume 4653 of *LNCS*, pages 874–883. Springer, 2007. Regensburg, Germany, September 3-7.

[7] M. d'Aquin, A. Schlicht, H. Stuckenschmidt, and M. Sabou. Criteria and evaluation for ontology modularization techniques. In H. Stuckenschmidt, C. Parent, and S. Spaccapietra, editors, *Modular Ontologies: Concepts, Theories and Techniques for Knowledge Modularization*, volume 5445 of *LNCS*, pages 67–89. Springer, 2009.

[8] C. Del Vescovo. The modular structure of an ontology: Atomic decomposition towards applications. In R. Rosati, S. Rudolph, and M. Zakharyaschev, editors, *24th International Workshop on Description Logics (DL 2011)*, volume 745 of *CEUR Workshop Proc.* CEUR-WS.org, 2011. Barcelona, Spain, July 13-16.

[9] C. Del Vescovo, D. Gessler, P. Klinov, B. Parsia, U. Sattler, T. Schneider, and A. Winget. Decomposition and Modular Structure of BioPortal Ontologies. In *10th International Semantic Web Conference (ISWC'11)*, volume 7031 of *LNCS*, pages 130–145. Springer, 2011. October 23-27, Bonn, Germany.

[10] P. Doran, I. Palmisano, and V. A. M. Tamma. SOMET: algorithm and tool for SPARQL based ontology module extraction. In *Third Workshop on Ontologies: Reasoning and Modularity (WoMO'08)*, volume 348 of *CEUR Workshop Proc.* CEUR-WS.org, 2008. Tenerife, Spain, June 2.

[11] A. C. Garcia, L. Tiveron, C. Justel, and M. C. Cavalcanti. Applying graph partitioning techniques to modularize large ontologies. In *Joint V Seminar on Ontology Research in Brazil and VII International Workshop on Metamodels, Ontologies and Semantic Technologies*, volume 938 of *CEUR Workshop Proc*, pages 72–83. CEUR-WS.org, 2012. Recife, Brazil, September 19-21.

[12] F. Giunchiglia, A. Villafiorita, and T. Walsh. Theories of abstraction. *AI Communication*, 10(3,4):167–176, 1997.

[13] J. Golbeck, G. Fragoso, F. W. Hartel, J. A. Hendler, J. Oberthaler, and B. Parsia. The national cancer institute's thesaurus and ontology. *Journal of Semantic Web*, 1(1):75–80, 2003.

[14] R. Hodgson and P. J. Keller. QUDT-quantities, units, dimensions and data types in OWL and XML. *Online (September 2011) http://www.qudt.org*, 2011.

[15] A. Kalyanpur, B. Parsia, E. Sirin, B. Cuenca Grau, and J. A. Hendler. Swoop: A Web Ontology Editing Browser. *Journal of Web Semantics*, 4(2):144–153, 2006.

[16] Y. Kazakov, M. Krötzsch, and F. Simancik. ELK reasoner: Architecture and evaluation. In *1st International Workshop on OWL Reasoner Evaluation (ORE-2012)*, volume 858 of *CEUR Workshop Proc.* CEUR-WS.org, 2012. Manchester, UK, July 1st.

[17] C. M. Keet. Using abstractions to facilitate management of large ORM models and ontologies. In R. Meersman, Z. Tari, P. Herrero, G. Méndez, L. Cavedon, D. B. Martin, A. Hinze, G. Buchanan, M. S. Pérez, V. Robles, J. Humble, A. Albani, J. L. G. Dietz, H. Panetto, M. Scannapieco, T. A. Halpin, P. Spyns, J. M. Zaha, E. Zimányi, E. Stefanakis, T. S. Dillon, L. Feng, M. Jarrar, J. Lehmann, A. de Moor, E. Duval, and L. Aroyo, editors, *OTM Workshops*, volume 3762 of *LNCS*, pages 603–612. Springer, 2005. Agia Napa, Cyprus, October 31 - November 4.

[18] C. M. Keet. Enhancing comprehension of ontologies and conceptual models through abstractions. In *10th Congress of the Italian Association for Artificial Intelligence (AI*IA'07)*, volume 4733 of *LNCS*, pages 813–821. Springer, 2007. Rome, Italy, September 10-13.

[19] C. M. Keet, C. d'Amato, Z. Khan, and A. Lawrynowicz. Exploring reasoning with the DMOP ontology. In *3rd Workshop on Ontology Reasoner Evaluation (ORE'14)*, CEUR Workshop Proc, pages 64–70. CEUR-WS.org, 2014. July 1, Vienna, Austria.

[20] Z. Khan and C. M. Keet. The foundational ontology library ROMULUS. In *3rd International Conference on Model & Data Engineering (MEDI'13)*, volume 8216 of *LNCS*. Springer, 2013. Sep 25-27, Amantea, Calabria, Italy.

[21] S. D. Larson, L. L. Fong, A. Gupta, C. Condit, W. J. Bug, and M. E. Martone. A Formal Ontology of Subcellular Neuroanatomy. *Front Neuroinformatics*, 1:3, 2007.

[22] N. F. Noy and M. A. Musen. Traversing ontologies to extract views. In *Modular Ontologies: Concepts, Theories and Techniques for Knowledge Modularization*, volume 5445 of *LNCS*, pages 245–260. Springer, 2009.

[23] C. Parent and S. Spaccapietra. An Overview of Modularity. In *Modular Ontologies*, volume 5445 of *LNCS*, pages 5–23. Springer Berlin Heidelberg, 2009.

[24] J. Pathak, T. M. Johnson, and C. G. Chute. Survey of modular ontology techniques and their applications in the biomedical domain. *Integrated Computer-Aided Engineering*, 16(3):225–242, 2009.

[25] H. Paulheim. On applying matching tools to large-scale ontologies. In *3rd International Workshop on Ontology Matching (OM'08)*, volume 431 of *CEUR Workshop Proc.* CEUR-WS.org, 2008. Karlsruhe, Germany, October 26.

[26] A. L. Rector, J. Rogers, P. E. Zanstra, and E. J. van der Haring. OpenGALEN: Open source medical terminology and tools. In *American Medical Informatics Association Annual Symposium (AMIA'03)*. AMIA, 2003. Washington, DC, USA, November 8-12.

[27] C. Rosse and J. L. V. Mejino. A reference ontology for biomedical informatics: the Foundational Model of A|atomy. *Journal of Biomedical Informatics*, 36(6):478–500, 2003.

[28] A. Schlicht and H. Stuckenschmidt. Towards structural criteria for ontology modularization. In P. Haase, V. Honavar, O. Kutz, Y. Sure, and A. Tamilin, editors, *1st International Workshop on Modular Ontologies (WoMO'06)*, volume 232 of *CEUR Workshop Proc.* CEUR-WS.org, 2006. November 5, Athens, Georgia, USA.

[29] A. Schlicht and H. Stuckenschmidt. A flexible partitioning tool for large ontologies. In *International Conference on Web Intelligence (WI'08)*, pages 482–488. IEEE Computer Society, 2008. 9-12 December, Sydney, NSW, Australia.

[30] J. Seidenberg. Web ontology segmentation: Extraction, transformation, evaluation. In *Modular Ontologies: Concepts, Theories and Techniques for Knowledge Modularization*, volume 5445 of *LNCS*, pages 211–243. Springer, 2009.

[31] V. Turlapati and S. Puligundla. Efficient module extraction for large ontologies. In *Knowledge Engineering and the Semantic Web*, volume 394 of *Communications in Computer and Information Science*, pages 162–176. Springer Berlin Heidelberg, 2013.

[32] M. Vigo, C. Jay, and R. Stevens. Design insights for the next wave ontology authoring tools. In *Conference on Human Factors in Computing Systems (CHI'14)*, pages 1555–1558. ACM, 2014. Toronto, ON, Canada, April 26 - May 01.

[33] P. L. Whetzel, N. F. Noy, N. H. Shah, P. R. Alexander, C. Nyulas, T. Tudorache, and M. A. Musen. BioPortal: enhanced functionality via new web services from the national center for biomedical ontology to access and use ontologies in software applications. *Nucleic Acids Research*, 39(Web-Server-Issue), 2011.